*Department of Electrical Engineering and Computer Science*

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

**6.858 Spring 2017**

# Quiz I

You have 80 minutes to answer the questions in this quiz. In order to receive credit you must answer the question as precisely as possible.

Some questions are harder than others, and some questions earn more points than others. You may want to skim them all through first, and attack them in the order that allows you to make the most progress.

If you find a question ambiguous, be sure to write down any assumptions you make. Be neat and legible. If we can't understand your answer, we can't give you credit!

Write your name and submission website email address on this cover sheet.

**This is an open book, open notes, open laptop exam.**
**NO INTERNET ACCESS OR OTHER COMMUNICATION.**

*Please do not write in the boxes below.*

| I (xx/16) | II (xx/10) | III (xx/12) | IV (xx/10) | V (xx/6) | VI (xx/10) | VII (xx/12) | VIII (xx/4) | Total (xx/80) |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

## Name:

## Submission website email address:

# I Paper reading questions

**1. [4 points]:**

Which of the following attacks are prevented in Google's design primarily by encrypting data at rest (on disk)?

**A.** Engineers gaining access to customer data.

**B.** Adversaries gaining access to customer data by physically stealing disks from a storage server.

**C.** Adversaries gaining access to customer data by exploiting vulnerabilities in application code such as the calendar service.

**D.** Adversaries passively monitoring network traffic between Google's data centers.

**2. [4 points]:**

Which of the following are reasonable arguments for deciding when to use phone-based two-factor authentication or a hardware token like RSA SecurID?

**(Circle True or False for each choice.)**

**A. True / False** The phone offers better usability, since the user need not carry a separate device.

**B. True / False** The phone offers significantly stronger security, since it has a more powerful processor that can compute a more CPU-intensive hash function.

**C. True / False** The phone offers significantly stronger security, since it can display longer one-time password codes (RSA SecurID displays just 6 digits).

**D. True / False** The phone offers weaker security, since the OS kernel may have bugs that allow other applications to tamper with the two-factor authenticator app.

**3. [4 points]:**

According to Paul Youn, if security was the absolute top goal for Airbnb, what should the company do?

**A.** Enable two-factor authentication for all customers.

**B.** Check the user's name against a valid credit card before opening an account.

**C.** Implement real-time fraud analytics.

**D.** Shut down the company's web site.

**4. [4 points]:**

Which of the following applications could be supported under Ryoan?

**(Circle True or False for each choice.)**

**A. True / False** Ryoan could support a video transcoding service, which returns a video re-encoded at a different resolution.

**B. True / False** Ryoan could support a tax preparation service which stored the user's tax data for past years to simplify filling in next years' taxes.

**C. True / False** Ryoan could support an encrypted email service that allowed users to send encrypted messages to one another.

**D. True / False** Ryoan could support a survey application, where each user submits his or her survey results to an application running on Ryoan, and at the end of the survey, the Ryoan application releases a summary of survey results.

# II Buffer overflows

Consider the following code snippet:

```
int foo(int x) {
  int a;
  int *p;
  void (*bar)(int);
  char buf[16];

  p = &a;
  bar = &somefunc;
  gets(buf);
  *p = 0;
  bar(a);
}
```

The stack layout of `foo` is as follows:

```
0x7fff0120: return address
0x7fff011c: saved ebp
0x7fff0118: variable a
0x7fff0114: variable p
0x7fff0110: variable bar
0x7fff0100: variable buf
```

Recall that `gets` does not stop at zero bytes.

There are several possible solutions, and you do not need to use every aspect of the code snippet in your solution.

**5. [10 points]:**

What input should an adversary provide to function `foo` to invoke `unlink("x.txt")`? Assume the code for `unlink` lives at address `0x40a01234`.

# III Baggy bounds checking

Consider the following code snippet:

```
struct foo {
  int a;
  int b;
  int c;
  char d[12];
};

struct foo *p;
p = malloc(sizeof(struct foo));

char *buf = p->d;
buf += N;
printf("0x%x\n", buf);
```

*N* is a constant integer value (positive or negative), between -1024 and 1024. Assume the program is running under Baggy bounds checking with `slot_size`=16, that the size of an `int` type is 4 bytes, and that the compiler does not insert any padding into the struct.

**6. [6 points]:**

Suppose $N = 18$, and `malloc()` returns `0x10203040`. What does the program print?

**7. [6 points]:**

The program prints `0x9001005a`. What was *N*? Hint 1: there is only one correct answer. Hint 2: recall that Baggy manipulates certain bits in pointers for bookkeeping.

# IV OKWS / lab 2

Ben Bitdiddle wants to implement HTTP request pipelining in zookws, to avoid the overhead of setting up a new TCP connection for each request. HTTP pipelining works as follows:

- When the client has multiple requests to the same server, it sends them back-to-back over the same TCP connection.
- The server sends responses to these requests in order, over the same TCP connection.
- The client uses the `Content-Length:` header in each response to determine when one response ends and the next response begins.

**8. [10 points]:**

Describe the changes that would have to be made to zookws to implement HTTP request pipelining. Your design should ensure that responses received by the client come from the correct service. That is, a compromised service should not be able to provide responses for requests to other services.

Hint: think about how a client determines where one response ends and the other begins, and think about what a malicious service might do to confuse the client. Once you add HTTP pipelining, can one service manipulate a connection also used by another service to send its response?

# V Trusted hardware

Intel SGX attestation for an enclave works by maintaining an append-only log of events since the creation of the enclave. The log receives 4 kinds of events, each corresponding to an instruction that the creator of the enclave can invoke:

- `ECREATE(start_address, end_address)` when the enclave is first created.
- `EADD(page_address)` when a page is added to the enclave. The contents of the page have already been initialized prior to `EADD` by the enclave's creator, but the contents of the page are separately attested to using the next instruction.
- `EEXTEND(address, data)` ensures that the 256-byte region of memory at `address` contains `data`. This allows the creator of an enclave to prove that the enclave's initial pages contain expected data.
- `EINIT` when the enclave has been fully initialized and cannot be changed from outside of the enclave.

Ben Bitdiddle writes some pseudocode to check an attestation generated from an Intel SGX enclave, as follows:

```
def verify(log, prog):
  ## prog holds the program we expect to be loaded
  if log[0] != { "opcode": "ECREATE", "create_start": prog.start, "create_end": prog.end }:
    return False
  if log[-1] != { "opcode": "EINIT" }:
    return False
  for logent in log[1:-1]:
    if logent["opcode"] == "EADD":
      if logent["add_start"] < prog.start || logent["add_end"] >= prog.end:
        return False
    elif logent["opcode"] == "EEXTEND":
      if prog.code[logent["extend_addr"] : logent["extend_addr"]+256] != logent["extend_data"]:
        return False
    else:
      return False
  return True
```

Assume that the contents of `log` are properly cryptographically authenticated, so that only a legitimate Intel SGX processor could have produced the log.

**9. [6 points]:**

Describe an attack by which an enclave creator can fool Ben's verifier into returning `True` but actually create an enclave that does not contain the expected program `prog`.

# VI Native Client

Ben Bitdiddle is implementing Software Fault Isolation for his Bentium processor, following the approach described in the Native Client paper. As far as the Native Client design is concerned, his processor is identical to the 32-bit x86 CPU considered by the Native Client designers.

Ben makes one mistake in his verifier: his verifier forgets to treat `nacljmp` as a single instruction during verification, and instead treats it as two instructions (`AND $0xffffffe0, %eax` and `JMP *%eax`) that comprise it.

For your reference, here are some Bentium instruction encodings (equivalent to 32-bit x86):

| Encoding | Instruction |
|---|---|
| `0f 05` | `SYSCALL` |
| `83 e0 e0` | `AND $0xffffffe0, %eax` |
| `ff e0` | `JMP *%eax` |
| `eb XX` | `JMP` to XX bytes past the start of the next instruction (i.e., XX=00 is effectively a no-op) |
| `b8 AA BB CC DD` | `MOV $0xDDCCBBAA, %eax` (load constant into %eax) |

**10. [10 points]:**

On the next page, write a program (i.e., the literal bytes that make up the program's executable code) that would pass Ben's verifier but that invokes the SYSCALL instruction at runtime. Assume the program starts executing at the beginning of your byte sequence, which gets loaded at the absolute address `0x00010000` (i.e., 64 KBytes). You should not need more than the 20 bytes that we have provided space for.

Here is an example of the syntax which you should follow in writing your answer:

**Syntax example:**

| Offset | Byte value | Instruction |
|---|---|---|
| 0x00 | 83 | `AND $0xffffffe0, %eax` |
| 0x01 | e0 | |
| 0x02 | e0 | |
| 0x03 | ff | `JMP *%eax` |
| 0x04 | e0 | |
| 0x05 | 0f | `SYSCALL` |
| 0x06 | 05 | |
| 0x07 | eb | `JMP` back to start of `SYSCALL` |
| 0x08 | fc | (4 bytes back; `0xfc` is -4) |

| Offset | Byte value | Instruction |
|---|---|---|
| 0x00 | | |
| 0x01 | | |
| 0x02 | | |
| 0x03 | | |
| 0x04 | | |
| 0x05 | | |
| 0x06 | | |
| 0x07 | | |
| 0x08 | | |
| 0x09 | | |
| 0x0a | | |
| 0x0b | | |
| 0x0c | | |
| 0x0d | | |
| 0x0e | | |
| 0x0f | | |
| 0x10 | | |
| 0x11 | | |
| 0x12 | | |
| 0x13 | | |
| 0x14 | | |

# VII Capsicum

Ben Bitdiddle wants to run a virus scanner on all of Ben's binary files in `/home/ben/bin` on his computer. Ben is worried the virus scanner might be malicious, and might try to read other files in Ben's home directory, or modify Ben's binary files. However, Ben is running FreeBSD on his computer, which supports Capsicum.

**11. [6 points]:**

What is the simplest way for Ben to isolate the virus scanner using Capsicum so as to prevent the virus scanner from accessing other files in Ben's home directory or modifying any of Ben's binary files? Be precise; describe both how the isolation is set up prior to running the virus scanner, and how the virus scanner must run once isolated.

Separately, the virus scanner developer is worried that the scanner code might be buggy, in that it might have buffer overflows that can be exploited when the virus scanner processes a specially crafted input file. This makes the virus scanner developer worried that, after the virus scanner encounters one specially crafted file, it might not report viruses it finds in other files.

**12. [6 points]:**

Describe how the virus scanner developer can avoid this problem using Capsicum. Even if one input file triggers a buffer overflow and arbitrary code execution in the virus scanner, the virus scanner should continue to scan other files. Your changes should not require changing your answer above; these changes should be implementable by the virus scanner developer without requiring the user (i.e., Ben) to do anything different.

## VIII 6.858

We'd like to hear your opinions about 6.858. Any answer, except no answer, will receive full credit.

**13. [2 points]:** Are there things you'd like to see improved in the second half of the semester?

**14. [2 points]:** Is there one paper out of the ones we have covered so far in 6.858 that you think we should definitely remove next year? If not, feel free to say that.

# End of Quiz